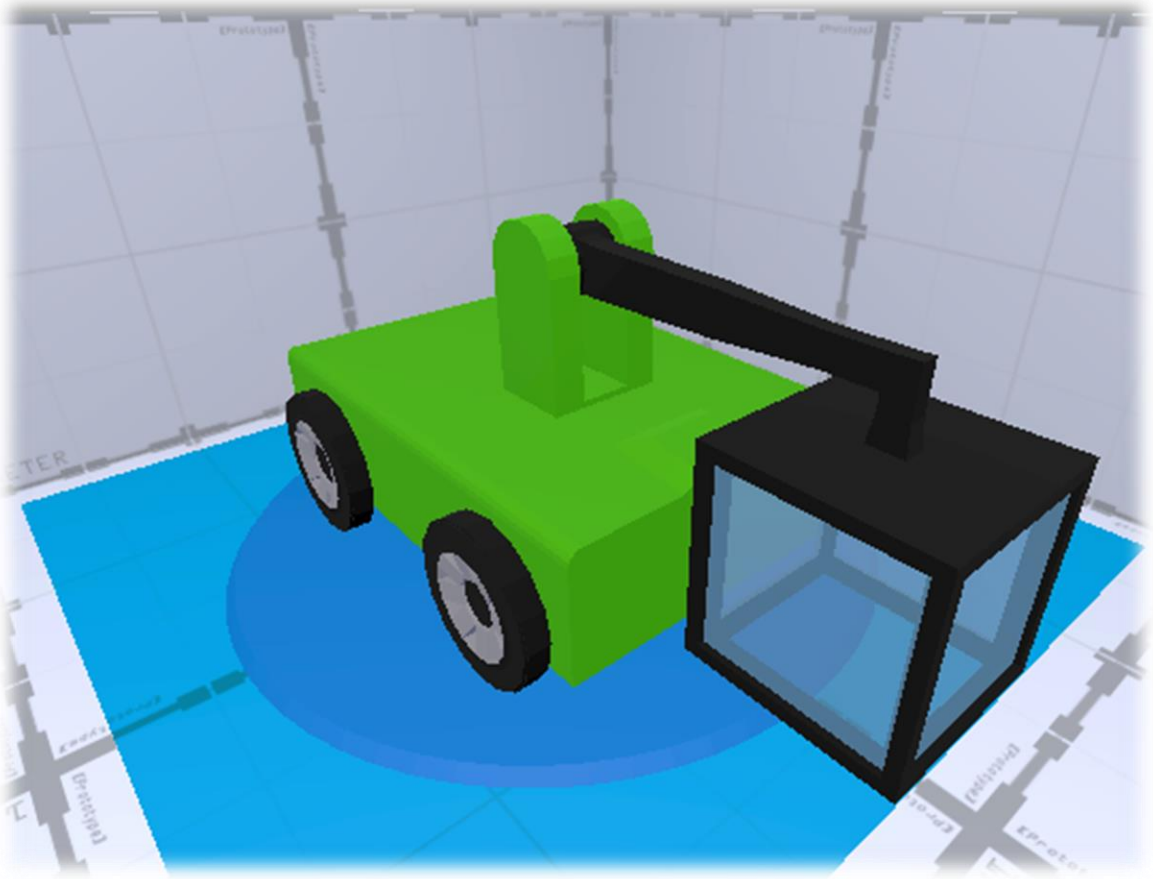
Documentação Peteca virtual

Documentação Peteca virtual



Peteca virtual é uma plataforma sendo desenvolvida pelo programa PET Desafios, do Grupo PET do curso de Engenharia de Controle e Automação do Instituto de Ciência e Tecnologia de Sorocaba.

Está em fase beta, com funcionalidades ainda sendo implementadas, bugs podem estar presentes.

Telas:

Nesta tela de **Início**, conforme a Figura 1, são apresentadas as **Notícias** sobre assuntos recentes pertinentes ao desenvolvimento da plataforma, novas implementações, resultados; Links de acesso a esta **documentação** e ao **site** do PET para mais informações; um **placar** com data, apelido e pontuação das pessoas que jogaram.

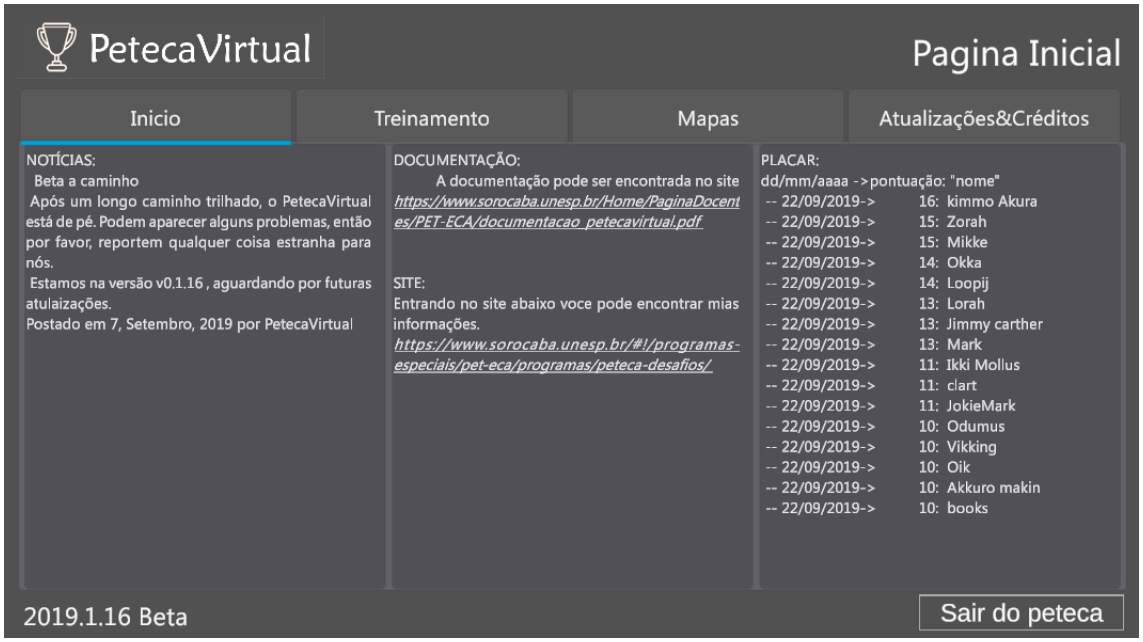


Figura 1- Imagem da tela de início da plataforma.

Na tela de **Treinamento** (Figura 2), são dispostos mapas para prática de comandos, e aprendizagem das mecânicas presentes. Para iniciar, seleciona-se um mapa, um modo de jogo e pressiona-se "jogar";

Com o mapa de treinamento for selecionado, há duas maneiras de marcar pontuação, **por contagem pontos**: que objetiva pegar o maior número de peças em um tempo determinado pelo mapa; ou **por contagem tempo**: que visa a coleta todos os itens no menor tempo;

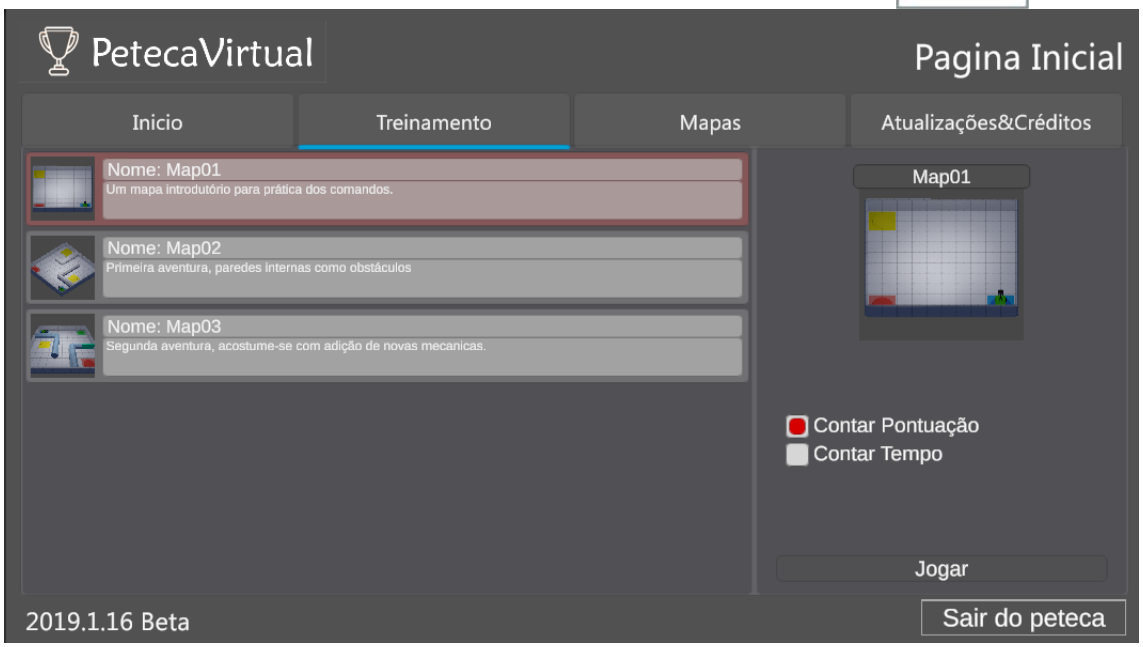


Figura 2- Imagem da tela de Treinamento da plataforma.

Na tela de **Mapas** (Figura 3), os verdadeiros desafios ainda estão para serem implementados, diferentemente do modo de treinamento, a marcação de pontuação dependerá de outros aspectos, nesta tela serão adicionados mapas com objetivos mais específicos a explorar comandos de programação.

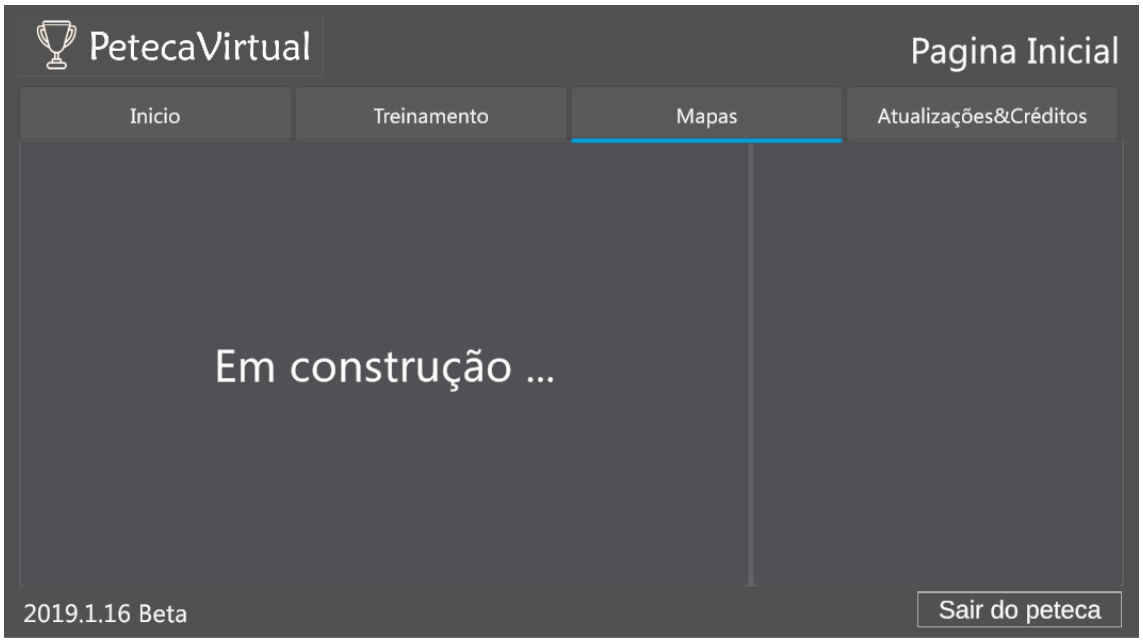


Figura 3-Imagem da tela Mapas da plataforma.

Na tela de **Atualizações e Créditos** (Figura 4), estão presentes os registros, que marcam os passos de avanço no desenvolvimento da plataforma; Uma área de atualização, que notifica caso novas versões estejam disponíveis , apresentando links para acesso a elas, e por fim uma área de créditos;

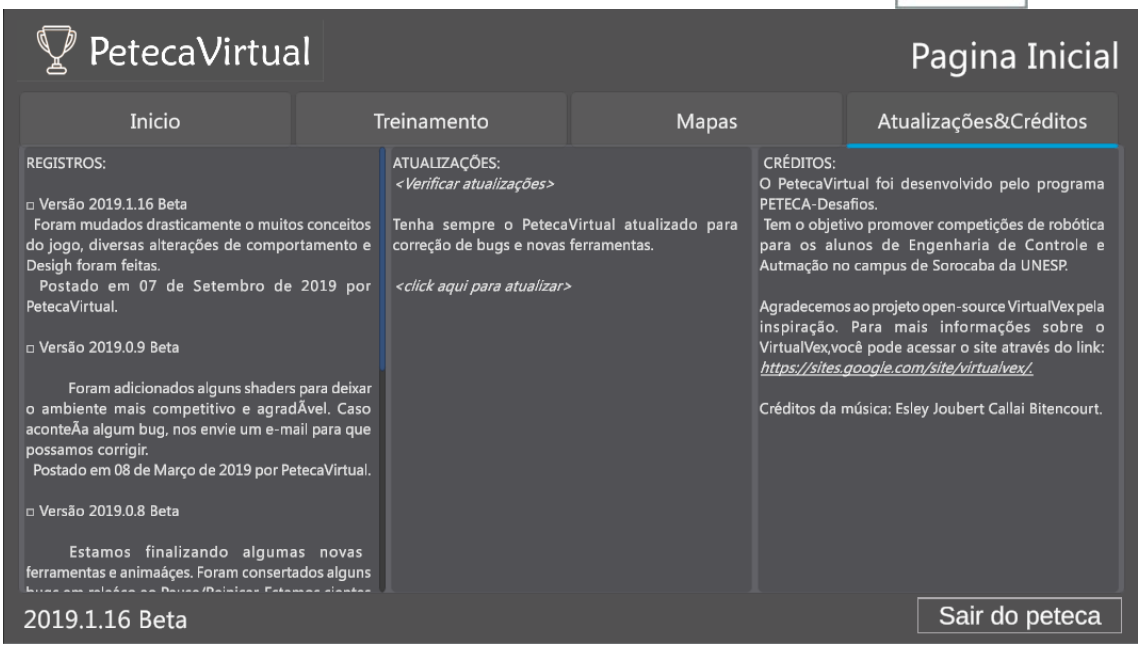


Figura 4- Imagem da tela de Atualizações e Créditos da plataforma.

Abrindo um mapa de treinamento:

Na tela de treinamento, após selecionar o mapa e o modo de jogo, inicia-se o controle indo em "Menu" e selecionando "Novo arquivo c++" (vide Figura 5). Um arquivo exemplo será aberto e nele ou em um novo arquivo (desde que tenha o "petecaVirtual.h" em sua pasta), o programa poderá ser escrito e copilado.

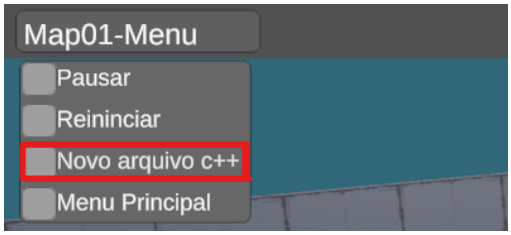


Figura 5-Menu para iniciar a programação do robô.

Controlando com aplicação em C++:

Cabeçalho:

Arquivo c++:

Como qualquer programa em c++ inicia-se importando bibliotecas necessárias e escrevendo na rotina principal: para que haja interação com a plataforma é necessário que importe a "**petecavirtual.h**", que deve estar presente na pasta do arquivo, a rotina principal é escrita em "**Int Main()**{}":

```
#include "petecavirtual.h"   //biblioteca necessária para interação com a plataforma
int Main(){    //Rotina principal
   /*
    *Seu programa vai aqui
   */
   System("pause");
   return 0;
}
```

Atributos DoLOOP e LOG:

Importando a biblioteca "*petecaVirtual.h*", existem duas *booleanas* DoLOOP e LOP, com `DoLOOP= true;` no final do programa c++, sua aplicação rodará novamente ao termino da programação pressionado enter, ao invés de fechar. Se tiver no início `LOG= true;` os comandos sendo enviados aparecerão no console.

Comandos de controle:

Básicos:

Os principais comandos para o controle do robô são ***Move***, ***Rotete*** e ***Claw***;

```
void Move(float v);
//Este comando tem como parâmetro uma float, que faz o
//robô andar "v" unidades na aplicação
/*Utilizando*/
    Move(10);
/*o robô anda 10 unidade na direção que estiver apontado*/
```

```
void Rotate(float a);
//Este comando tem como parâmetro uma float, que faz o
//robô girar "a" graus no sentido horário na aplicação
/* Utilizando */
    Rotate(-90);
/*o robô gira 90 graus no sentido anti-horário*/
```

```
void Claw();
//Este comando não tem parâmetros
//alterna o estado da garra do robô
/*Utililzando*/
    Claw();
//a garra abaixa se estiver levantada, ou, levanta se abaixada
```

Ambiente:

Os principais comandos para o controle do ambiente são ***CameraSel*** e ***Restart***;

```
void CameraSel(int cam);
//Este comando tem como parâmetro um inteiro, que
//seleciona entre as cameras disponíveis
/*Utililzando*/
    CameraSel(1);
//a camera é alternada para a camera de índice 1
```

```
void Restart();
//Este comando não tem parâmetros
//Reinicia o mapa aberto, assim como sua contagem de tempo
/*Utililza-se*/
    Restart();
```

De tempo:

Os principais comandos para o controle de tempo são ***GameSpeed*** , ***Wait*** e ***RealWait***;

```
void GameSpeed(float v);
//Este comando tem com parâmetro uma float, que
//altera a velocidade de execução do jogo
/*Utililzando*/
    GameSpeed(0.5);
//faz com que as ações do jogo aconteça a 50% da velocidade padrão,
//o tempo do jogo é escalonado junto
```

```
void Wait(float t);
//Este comando tem com parâmetro uma float, que
//segura a execução das ações por t segundos , na escala de tempo do jogo
/*Utililzando*/
    Wait(5);
//ocorre uma pausa de execução das açoes de 5 segundos
//Alternativamente, pode-se usar
    RealWait(5);
//que segurará as ações por 5 segundos do tempo real, independentemente se
//a aplicação estiver pausada ou da escala de tempo do jogo
```

Com Resposta:

Os principais comandos para o controle de tempo são ***GetTime***, ***GetCameraCount*** e ***TestSensor***;

Esses comandos, diferentemente dos anteriores, recebem valores da plataforma, interrompendo a fila de comandos, até que sejam executados, e retornam valores para o programa em C++.

```
float GetTime()
//Este comando não tem parâmetros, e retorna em formato de uma float
// o tempo do jogo
/*Utilizando*/
    float v;
    v= GetTime();
//o valor do tempo é passado para a variável "v", podendo ser
//posteriormente utilizada na lógica implementada
```

```
int GetCameraCount()
```

```
//Este comando não tem parâmetros, e retorna em formato de uma int
//o numero de cameras presentes no mapa
/*Utilizando*/
    float nCameras;
    nCameras= GetCameraCount();
//A quantia de cameras é passado para a variável "nCameras", podendo ser
//posteriormente utilizada para,  por exemplo, saber até que camera
//pode ser selecionada com CameraSel(i)
```

No Robô estão pressentes 6 sensores, como mostra a Figura 6 a seguir, são respectivamente, de indice (0 ao 5) Frontal(F), Frontal_direito(FR), Frontal_esquerdo(FL), Trazeiro(R), Trazeiro_direito(RR), Trazeiro_esquerdo(RL).

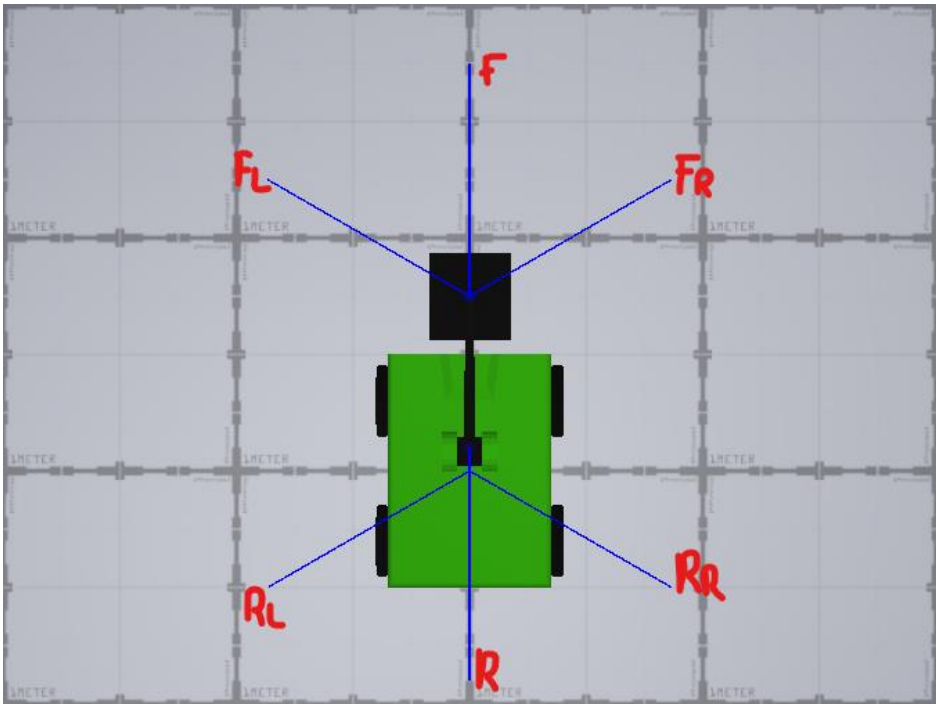


Figura 6-Representação dos sensores do Robô.

Esses sensores são robustos e invisíveis no jogo, retornam 1 ou 0 se estiverem ou não tocando em uma parede. Com a função **TestSensor**(*num*), podem ser acessados de acordo com seu índice, testando um sensor com índice maior do que o inexistente sempre terá como resultado "-1"

```
int TestSensor(int i)
//Este comando tem com parâmetro o indice do sensor a ser testado,
// e retorna em formato de uma int
// 0 se não estiver detectando uma parede
// 1 se estiver detectando uma parede
//-1 se o indice levar a um sensor inexistente
/*Utilizando*/
    if(TestSensor(0)==0){
        cout<<"Frente livre";
        /*faça algo*/
    }
//O sensor frontal é testado, se não houver parede na frente, os comandos
//do bloco abaixo é executado
```